# Тестируем группировки сервисов. На узле разработчика

Как поведет себя система, когда сеть тормозит, сервис в цепочке сломали, подсеть отвалилась?

Якушкин Олег





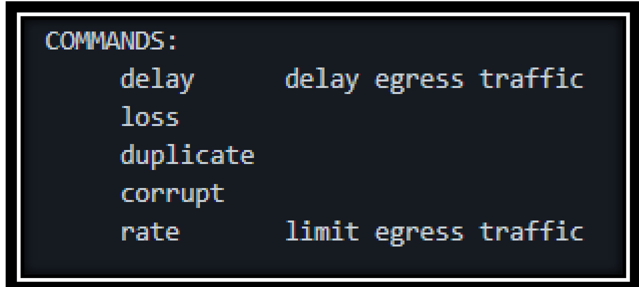

# Быстрый старт?

**CHAOS TESTING**

Требует:

- Observability
- Поднятых контейнеров

Может:



```
COMMANDS:
     delay      delay egress traffic
     loss
     duplicate
     corrupt
     rate       limit egress traffic
```






лимитирован только работой с сетью,
**хочется управления узлами** и создавать сценарии





# Чего же мы хотим?

Управлять сетью, контролировать запуск узлов

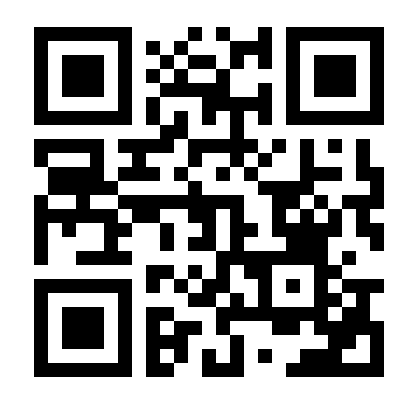
## SDN как база

**Инструмент, способный:**

- настраивать изолированные сервисы
- создавать сервисам проблемы
- наблюдать реакции









# L3NS

## Максимально простой инструмент

- Создание узлов из докер-образов
- Развертка простых сетей
- Доступ к отправке команд на узел





```python
from l3ns.ldc import DockerNode
from l3ns import defaults

n1 = DockerNode('test1', image='alpine', command='tail -f /dev/null')
n2 = DockerNode('test2', image='alpine', command='tail -f /dev/null')

n1.connect_to(n2)

print(n1.get_ip())
print(n2.get_ip())

defaults.network.start(interactive=True)
```





# Управлять узлами

## По ssh

- Создадим докер-образ
- Разрешим ssh
- Запустим демона
- Передадим команды!

# Подготовка зависимостей

```
# docker build -t sshnode .
FROM ubuntu:20.04

## kubenode software base
RUN apt update \
    && apt install  -yq  software-properties-common  ca-certificates openssh-client  apt-transport-https \
    wget curl nano htop iptables supervisor systemd

### SSH
RUN useradd --create-home --no-log-init --shell /bin/bash -g root vagrant && \
    usermod -aG sudo vagrant && \
    usermod -aG users vagrant && \
    echo "vagrant:vagrant" | chpasswd
```



# Подготовка ssh

```
RUN apt update && apt --no-install-recommends install -y putty-tools plink openssh-server ssh && \
    mkdir /var/run/sshd && \
    sed -ri 's/^#?PermitRootLogn\s+.*/PermitRootLogin yes/' /etc/ssh/sshd_config && \
    sed -ri 's/UsePAM yes/#UsePAM yes/g' /etc/ssh/sshd_config && \
    echo "\nAllowGroups root" >> /etc/ssh/sshd_config && \
    echo "\nAllowUsers vagrant" >> /etc/ssh/sshd_config && \
    mkdir /root/.ssh && \
    mkdir /home/vagrant/.ssh
EXPOSE 22
```



# Создание сети

```
defaults.network = Network('49.0.0.0/8')

master = DockerNode('master', image='sshnode', command='tail -f /dev/null', tty=True, stdin_open=True, )
worker1 = DockerNode('worker1', image='sshnode', command='tail -f /dev/null', tty=True, stdin_open=True)
worker2 = DockerNode('worker2', image='sshnode', command='tail -f /dev/null', tty=True, stdin_open=True)

master.connect_to(worker1)
master.connect_to(worker2)
master.connect_to_internet = True
worker1.connect_to_internet = True
worker2.connect_to_internet = True
```



# Запуск сети и общение в ней

С мастера заходим на каждый из узлов

```
with defaults.network:
    run_on_all_nodes( """/bin/bash -c "/usr/sbin/sshd" """)
    print("started SSH deamons on all nodes")
    run_on_a_node(master, """/bin/bash -c "echo y | plink -ssh vagrant@"""+ \
                  str(worker1.get_ip())+ """ -pw vagrant \"exit\" " """)
    run_on_a_node(master, """/bin/bash -c "plink -ssh vagrant@"""+ \
                  str(worker1.get_ip())+ """ -pw vagrant ls -lah" """)
```





# Ограничивать сеть

## Используя tc



- Установить в образ tc
- Запускать с «привилегированным» флагом
- Передавать tc-команды

```
tc qdisc add dev eth0 root netem delay 200ms
```

- **qdisc**: изменять планировщик
- **add**: добавить новое правило
- **dev eth0**: по отношению к устройству eth0
- **netem**: использовать эмулятор сети
- **delay**: изменяемое свойство – задержка
- **200ms**: добавить задержку 200мс







# Использовать несколько узлов

- Применить Docker Swarm
- Использовать ее как точку входа и организации сети

# Запускаем узлы удаленно

## В Docker Swarm

```
from l3ns.swarm import SwarmNode, SwarmSubnet, SwarmHost
from l3ns import defaults
from l3ns.base import Network

defaults.network = Network('15.0.0.0/8')
host1 = SwarmHost('w1', login='ojakushkin_w1')
host2 = SwarmHost('w3', login='ojakushkin_w3')
n1 = SwarmNode('test1', host1, image='alpine', command='tail -f /dev/null')
n2 = SwarmNode('test2', host2, image='alpine', command='tail -f /dev/null')

n1.connect_to(n2, subnet_class=SwarmSubnet)
defaults.network.start(interactive=True)
```





# Организовать кластер

- Создать Swarm-сеть
- Напихать в нее роутеров и узлов
- Организовать сетевые связи





# Кластер «на коленке»

## Формируем узлы, связи, сеть

```
for i in range(m):
    ii = i // cluster_size
    lan_nodes = []
    lans.append(lan_nodes)
    lan_routers = SwarmNode.generate_routers(f'l3ns_router{i}_', inteerconnect, hosts[ii])
    routers += lan_routers
    for r in lan_routers:
        router_net.add_node(r)
    lan_nodes.extend(lan_routers)
    lan_nodes.extend([
        SwarmNode(f'l3ns_node_{i}_{j+1}', hosts[ii], image='alpine',
                  command=f'ping {random.choice(random.choice(lans)).get_ip()}')
        for j in range(N)])
    lan_net = DockerSubnet(f'l3ns_net_{i}', lan_nodes, docker_client=hosts[ii].get_docker_client())
overlay = RipOverlay(routers)
```

HL HighLoad++

# MADT ~= L3NS

- Эмуляция L3
- Quagga (SDN) вместо эмуляции роутеров
- Docker-контейнеры для эмуляции хостов
- Python API для определения топологии сети
- Базовая настройка динамической маршрутизации из коробки



## large_net containers:

back to list | open graph | restart | stop

- Uptime: 112 sec.
- Avg. traffic: 0.24621058688614814
- Messages: 13583
- 3: 0.8% | 103

Show containers

back to list | open graph | restart | stop

- Uptime: 102 sec.
- Avg. traffic: 0.21274555447195814
- Messages: 12335
- 3: 0.8% | 103

Filter

MADT_large_net_L2_N25_5 (madt/pyget:latest)

MADT_large_net_L2_N25_4 (madt/pyget:latest)

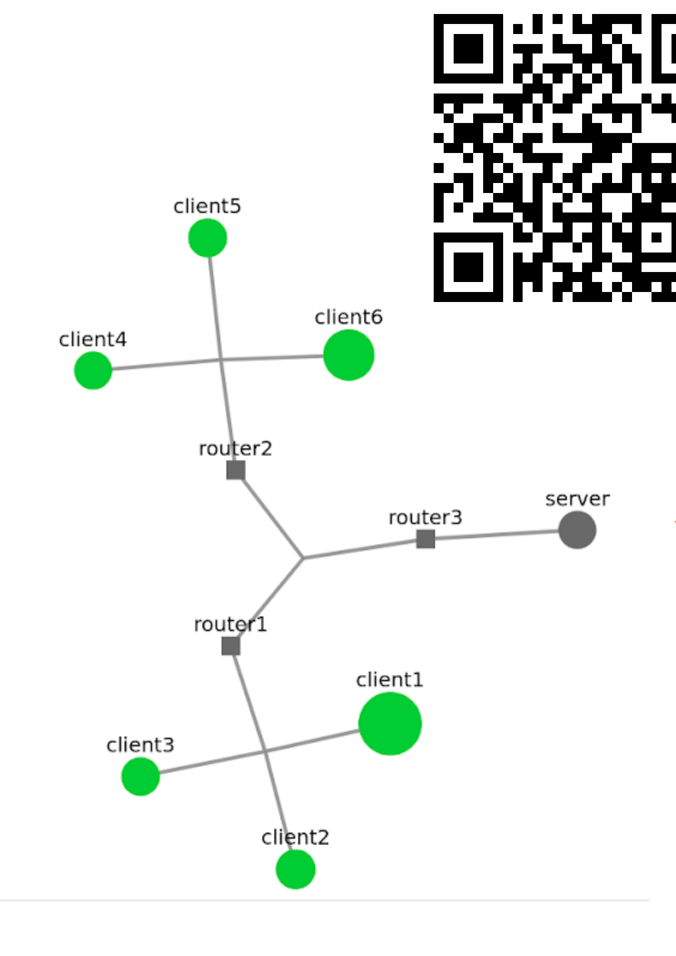
MADT_large_net_L2_N25_3 (madt/pyget:latest)

MADT_large_net_L2_N25_2 (madt/pyget:latest)

MADT_large_net_L2_N25_1 (madt/pyget:latest)

MADT_large_net_L2_N24_5 (madt/pyget:latest)






# Отслеживать все пакеты!

- Протокол OSPF
- Роутеры – docker-узлы
- Использовать сниффер tshark
- Логировать по чек-сумме и пункту назначения на каждом узле
- Агрегировать





# Вывод tshark

```
    Destination: 15.0.0.3
ransmission Control Protocol, Src Port: 36546 (36546), Dst Port: 8
    Source Port: 36546
    Destination Port: 80
    [Stream index: 59]
    [TCP Segment Len: 0]
    Sequence number: 141    (relative sequence number)
    Acknowledgment number: 853    (relative ack number)
    Header Length: 32 bytes
    Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: *******A****]
    Window size value: 501
    [Calculated window size: 64128]
    [Window size scaling factor: 128]
    Checksum: 0x1e45 [validation disabled]
        [Good Checksum: False]
        [Bad Checksum: False]
```

HighLoad++


















# Моделировать распределенные реестры

- Алгоритмы консенсуса
- Поведение при непогоде в сети





NEO
smart economy

ethereum

r3.corda

HYPERLEDGER
FABRIC

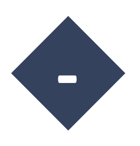
Tendermint



# MadT




Топология строится до запуска




Нет АПИ для динамического взаимодействия




Есть пользовательский интерфейс для наглядной работы с запущенной сетью



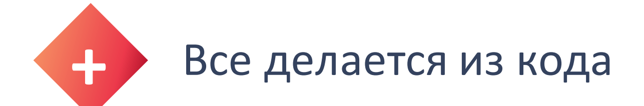


# L3NS

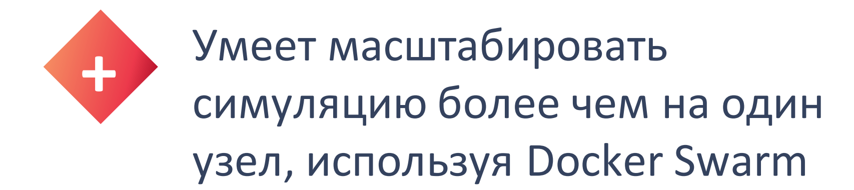

-
Нет графического интерфейса


-
Минималистичен
выше меры


-
Зелен


+
Все делается из кода


+
Запускаем с WSL2 на windows


+
Умеет масштабировать
симуляцию более чем на один
узел, используя Docker Swarm




HL
HighLoad++
Весна 2021

# Какие есть альтернативы для этих задач?

1 **Mininet, NS-3, CoreEmu – о них чуть позже**

2 **EVE-NG - платный**

3 **GNS3 – фокус именно на сети**





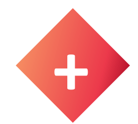
# Mininet




Эмуляция
L2 (OpenVSwitch)




Изначально для эмуляции хостов используются виртуальные машины




Но есть плагины, самые интересные:

- ContainerNet — контейнеры для эмуляции хостов
- MaxiNet — распределённое тестирование




Python API для описания топологии сетей




Продвинутый CLI управления моделями и запуска тестов

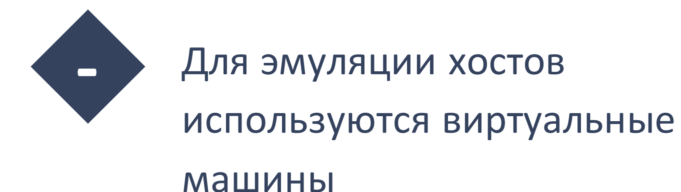



Библиотека топологий, в том числе и параметризированных



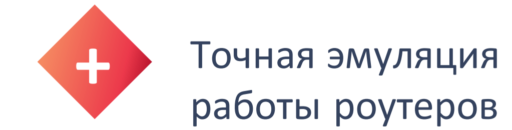


# EVE-NG




- Структура сети описывается графически, нет API
- Ручная настройка протоколов динамической маршрутизации

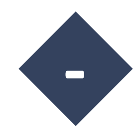







# GNS3




Ручная настройка динамической маршрутизации




Точная эмуляция работы роутеров




HTTP API для определения топологии сети




Также эмуляция L2




Можно использовать Docker-контейнеры



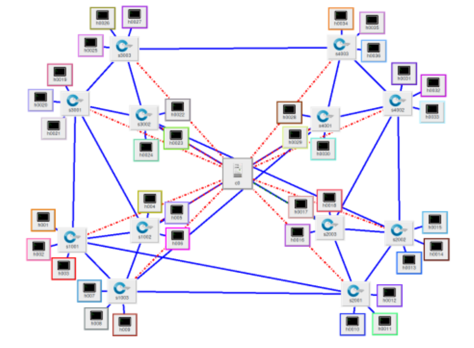


# Интересные дополнительные кейсы

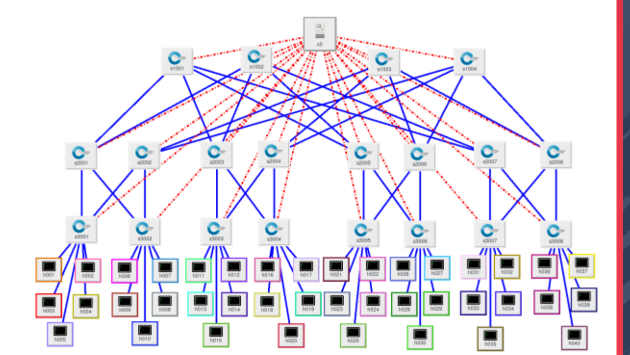
1 **MPI**

2 **Kubernetes**

3 **Потери в беспроводной мобильной связи**







# Кластер с MPI

- Используем Containers Net на базе mininet
- Строим топологии Fat-Tree и Dragonfly











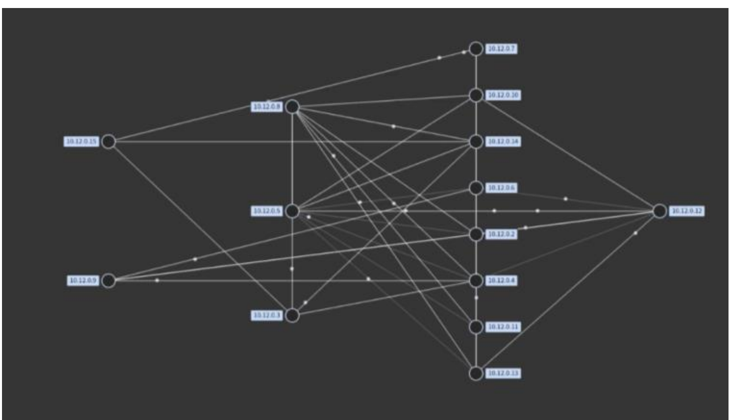

# Kubernetes

- Используем Istio
- Мы все перепробовали, но запустить Istio + Kubernetes на одной машине не удалось=(



# Моделирование беспроводного соединения








- NS-3 + Docker для моделирования
- Vizceral для визуализации

- Альтернативно можно применить coreemu, у которого есть обертки над NS-3





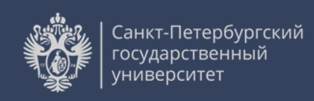
# Выводы

## И проблемы

| платформа | база эмуляции сети | поддержка запуска контейнеров | графический интерфейс симуляции | поддержка WSL2 | распределенная работа |
|---|---|---|---|---|---|
| coreemu | + | + | + | | + |
| l3ns | + | + | | + | + |
| madt | + | + | + | | |
| ns-3 | + | не офф | | | + |
| mininet | + | не офф | + | | не офф |











# Спасибо за внимание!

## Дискуссия

**Тестируем группировки сервисов. На узле разработчика**
Как поведет себя система, когда сеть тормозит,
сервис в цепочке сломали, подсеть отвалилась?

Олег Якушкин и команда СПбГУ